

Schnell und kraftvoll

CVS-Alternativen bietet auch das kommerzielle Lager: Für Perforce sprechen nicht nur Feature-Liste und Support, sondern auch, dass freie Softwareprojekte eine kostenfreie Lizenz bekommen. Arnt Gulbrandsen



Wer umfangreiche und sich rasch weiterentwickelnde Softwareprojekte mit CVS verwaltet, stößt früher oder später an die Grenzen des klassischen Versionsmanagement-Werkzeugs. Die Verwaltung von Branches – essenziell in allen Projekten, die auch für ältere Versionen der eigenen Software noch Bugfixes bereitstellen – ist eine Qual, die Netzwerk-Performance schlecht und bei unbilligen Forderungen wie dem Versionieren von JPEG-Dateien oder in einem »chroot«-Gefängnis eingesperrt verhält es sich undokumentiert merkwürdig.

Alle Fixes aus einem Bugfix-Branch inkrementell in den Hauptbranch übernehmen, dabei aber auf das Kopieren diverser Workarounds verzichten – das erweist sich als echte Herausforderung für einen Release-Manager. Die Dauer eines »cvs update«, selbst wenn sich nur eine einzige Datei geändert hat, verhält sich proportional zur Gesamtanzahl der Dateien im Repository. Und welcher CVS-User hat sich nicht schon darüber geärgert, dass der Source

Code Manager (SCM) Konfliktmarkierungen ohne Vorwarnung über die diversen Dateien verteilt? Konfliktmarker sind ohne Zweifel sehr nützlich, aber warum kann CVS nicht erst einmal jenen Teil des Codes auf den neuesten Stand bringen, an dem man gerade nicht arbeitet, und sich um den Rest später kümmern?

Die Alternative

Wenn der Leidensdruck stark genug wird, macht sich so mancher auf die Suche nach einer Alternative. Die heißt zum Beispiel Perforce. Oberflächlich gesehen weist dieses Versionskontrollsystem genügend Ähnlichkeiten mit CVS auf, sodass der Umstieg nicht allzu schwer fällt. Ob man zum Einchecken ins Repository, das jetzt Depot heißt, »p4 submit« statt »cvs commit« sagt, macht vom Gefühl her keinen großen Unterschied. Emacs-Usern dürfte es ebenso egal sein, ob das entsprechende Modul zum Einbinden des Clients den Namen

»p4.el« statt »vc.el« trägt.

Im Hintergrund jedoch überwiegen die Unterschiede. Der Perforce-Server weiß viel, viel mehr als sein CVS-Äquivalent. Alle Daten, die CVS in »CVS/*« ablegt, verwaltet der »p4d« in einer eigenen Datenbank. Auch das Client-Server-Protokoll arbeitet sehr viel effizienter. Das hat Folgen: »p4 sync« bringt die lokale Arbeitskopie des Repository unvergleichlich schneller auf den aktuellen Stand, als der entsprechende CVS-Befehl »cvs update« es kann.

Auch die Branching-Systeme der beiden SCMs unterscheiden sich. CVS ordnet die Informationen zu den Branches den Dateien zu, Perforce dagegen dem jeweiligen Verzeichnis. In CVS kann eine Datei mehrere Branches enthalten; in Perforce gehört jede Datei zu einem Zweig (Verzeichnis) und die Verzeichnisse sind durch Branching miteinander verbunden. (Die volle Wahrheit ist leider etwas komplexer – doch das führte in diesem Artikel zu weit.) ▶

Perforce

Homepage: [<http://www.perforce.com/>]

Download: [<http://www.perforce.com/perforce/loadprog.html>]

Preise: 750 US-Dollar pro Benutzer im ersten Jahr, anschließend 150 US-Dollar pro Benutzer; Rabatte für Großkunden, kostenlos für freie Softwareprojekte

Unterstützte Plattformen: (Client und Server):

Linux: i386, Alpha, Sparc, PowerPC, Mips, IA64, S390

Andere Unix-Systeme: FreeBSD, Solaris, Mac OS X und zirka weitere 20

Sonstige: Mac OS 8.5, Windows NT/XP/2000, Amiga OS, OS/2, Be OS, VMS und etwa zehn weitere

Einfach und effektiv – so lässt sich das Zugriffskontrollsystem von Perforce beschreiben. Anders als bei CVS sind atomare Checkins ebenso selbstverständlich wie die Tatsache, dass Perforce beim Abgleichen der Arbeitskopie mit dem Depotinhalt durch »p4 sync« niemals Dateien beschädigt, an denen der User gerade arbeitet.

Die Firma

Hinter dem Hersteller Perforce Software verbirgt sich eine Firma mit 60 Angestellten, die außer dem SCM nichts anderes herstellt. Open-Source-Projekte bekommen kostenlose Nutzungslizenzen. Anders als bei vielen anderen Softwareherstellern gehört eine ordentliche Qualitätsgarantie zum Inhalt des Lizenzvertrags. Sollte der SCM jemals etwas zerstören, übernimmt Perforce Software die Verantwortung dafür. Nicht zuletzt ist exzellenter Support im Lizenzpreis enthalten. Zu den Projekten, die Perforce als SCM einsetzen, zählen QT, Perl und Unreal Tournament 2003.

Das Client-Programm »p4« und der Server »p4d« – passend zu dem gewünschten Betriebssystem – liegen unter [\[ftp://ftp.perforce.com/\]](ftp://ftp.perforce.com/) (alternative URL siehe **Kasten „Perforce“**) zum Download bereit.

Erste Schritte

Auch ohne Lizenzschlüssel lässt sich damit bereits eine voll funktionsfähige Installation für zwei Benutzer und zwei

Listing 1: Client-spezifischer Blick aufs Depot

```

01 ...
02 # Use 'p4 help client' to see more about client views
   and options.
03 ...
04
05 Root:   /home/billg/work/projectx
06
07 Options:      noallwrite noclobber compress crlf
   unlocked nomodtime rmdir
08
09 View:
10     //depot/projectx/src/main/... devel/...
11     //depot/projectx/vendor/... vendor/...
12     //depot/projectx/re1/1.0/... re1.0/...

```

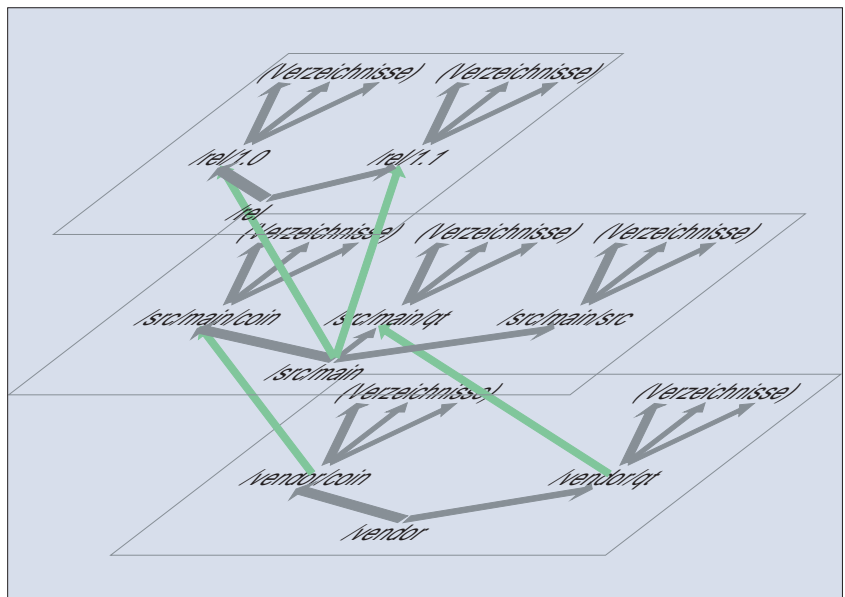


Abbildung 1: Die Ebenen dieser 3D-Grafik zeigen Release-, Entwicklungs- und Vendor-Branch, grün dargestellt sind die Integrationen.

Client-Verzeichnisse (zum Beispiel in »/tmp«) aufsetzen:

```
mkdir /tmp/p4-examples
p4d -r /tmp/p4-examples &
```

Anschließend gilt es, dem Client zu erklären, dass er den neu gestarteten Server nutzen soll:

```
mkdir projectx
echo P4PORT="/bin/hostname":1666 > 2
projectx/.p4config
export P4CONFIG=.p4config
```

Die Umgebungsvariable »P4CONFIG« teilt dem Client-Programm »p4« den Namen seiner Konfigurationsdatei (im Beispiel »p4config«) mit. Einen Pfad muss man nicht angeben: »p4« sucht im aktuellen Arbeitsverzeichnis, dem darüber liegenden Ordner und gegebenenfalls weiteren Elternverzeichnissen so lange nach »p4config«, bis es fündig wird. Dann nutzt der Client den darin verzeichneten Perforce-Server.

Bei großen und komplexen Projekten arbeitet normalerweise eine Reihe von Leuten an verschiedenen Versionen des Codes. Werden Branches genutzt, gibt es einen oder mehrere Entwicklungszweige, einen für die Qualitätssicherung (QA, Quality Assurance), einen für jede Release, oft auch noch einen Vendor-Branch (in dem die Originale des ver-

wendeten Third-Party-Codes lagern) und weitere nach Bedarf.

Angenommen es handelt sich um eine große GUI-Applikation auf der Basis von QT und Coin. In diesem Fall muss sich der Buildmaster des Projekts (also die Person, in deren Händen die Verwaltung des Codes in letzter Instanz liegt) sowohl um den in der Entwicklung befindlichen Code, den QA- beziehungsweise den für Releases vorgesehenen Code als auch um den Vendor-Code kümmern:

```
$ ls ~/work/projectx
devel  re1.0  vendor
$ ls ~/work/projectx/vendor
coin  qt
$ ls ~/work/projectx/devel
coin  src  qt
```

Im »vendor«-Verzeichnis befinden sich separate Ordner für QT und Coin, in »devel« liegen alle Dateien, an denen die Entwickler arbeiten, inklusive QT und Coin. Zwei Versionen von QT, zwei von Coin – warum das?

»vendor/coin« enthält die Originalausgabe von Coin, »devel/coin« die an die eigenen Erfordernisse angepasste und tatsächlich verwendete Version. Idealerweise sind beide zu 100 Prozent identisch (wenn das der Fall ist, sorgt Perforce dafür, dass sie auch nur einmal gespeichert werden), aber nicht notwendigerweise. Womöglich enthält »devel/

coin« eine ältere Version als »vendor/coin« oder lokale Anpassungen.

In Perforce ist »devel/coin« ein Branch von »vendor/coin«. Änderungen werden in einen Branch eingechekkt und optional in den anderen integriert. Wenn eine neue Coin-Version herauskommt, entpackt sie der Buildmaster in »vendor/coin«, checkt sie in Perforce ein und integriert sie mit »p4 integrate« in »devel/coin«. Die anderen Programmierer benutzen ausschließlich die Version aus »devel/coin«.

»p4 integrate« bedeutet, das Perforce alles Neue aus »vendor/coin« nach »devel/coin« überträgt. Alle neuen Vendor-Änderungen (aus »vendor/coin«) werden in die Produktionsausgabe in »devel/coin« übernommen. Das Ergebnis legt Perforce in »devel/coin« ab. Die Schwierigkeit ist, dass nur Änderungen in Betracht kommen, die seit der letzten Integration hinzukamen.

Glücklicherweise übernimmt Perforce den größten Teil dieser komplizierten und langweiligen Aufgabe automatisch mit »p4 resolve -am« (Automatic Merge). Der Buildmaster muss sich eventuell um den einen oder anderen Konflikt kümmern, gegebenenfalls (etwa bei Binärdateien wie JPEGs oder eigenen Dateiformaten wie für Game-Level-Dateien) unterstützt durch ein eigens zu diesem Zweck geschriebenes oder zugekauftes Merger-Tool.

Nach der Integration kompiliert und testet der Buildmaster das Ergebnis und überträgt all diese Änderungen zum Schluss mit »p4 submit devel/coin/...« ins Depot. Wenn die anderen Programmierer das nächste Mal »p4 sync« aufrufen, bringt Perforce deren Coin-Kopien auf den neuesten Stand.

Eine Frage der Ansicht

Anders als die übrigen Mitarbeiter am Projekt sollte der Buildmaster einen möglichst umfassenden Überblick über den Code haben. Entsprechend sieht sein Client alles. Die Definition der Buildmaster-Ansicht zeigt [Listing 1](#).

Einen solchen Client-spezifischen Blick aufs Depot ändert man mit »p4 client«. Auf diesen Befehl hin lädt »p4« die ak-

tuelle Definition zur Korrektur in den in der Umgebungsvariablen »EDITOR« festgelegten Editor. Zum reinen Einsehen der Spezifikation dient der Perforce-Befehl »p4 client -o«.

Die Variablen »P4CLIENT« und »P4USER« legen den Perforce-Client- und Benutzernamen in ».p4config« oder der Shell-Umgebung fest. Die Defaultwerte (also der Rechnername für »P4CLIENT«, der Unix-Username für »P4USER«) decken den Normalfall ab, bei dem alle der Versionskontrolle unterliegenden Daten unterhalb ein und desselben Wurzelverzeichnis liegen. Dies ist für den jeweiligen Client hinter dem Schlagwort »Root:« zu finden. »View:« bestimmt dann noch, welche Teile des Depots dieser Client zu sehen bekommt: Die Definition in [Listing 1](#) macht drei unterschiedliche Depotbestandteile in drei Unterverzeichnissen von »/home/billg/work/projectx« sichtbar.

»~ billg/work/projectx/vendor« spiegelt alles, was Perforce in »//depot/projectx/vendor« enthält; die drei Punkte »...« stehen für „alle Dateien und Unterverzeichnisse“. »~ billg/work/projectx/devel« übernimmt die Daten aus »//depot/projectx/src/main« und in »~ billg/work/projectx/rel1.0« landet alles, was Perforce unterhalb von »//depot/projectx/rel/1.0« verwaltet. Die Client-Ansicht eines einfachen Programmierers umfasst in den meisten Fällen nur einen Ausschnitt des Depots ([Listing 2](#)).

Kontrolle ist besser

Natürlich erlaubt Perforce dabei auch Zugriffskontrollen. Soll nur der Buildmaster Schreibzugriff auf »//depot/projectx/vendor« haben und die Programmierer lediglich »//depot/projectx/src/main« verändern dürfen, packt man Letztere mit »p4 group« in eine Gruppe und weist ihr mit »p4 protect« die entsprechenden Rechte zu (siehe [Listing 3](#)).

Die zweite Zeile besagt, dass alle »list«-Rechte für alles haben. Sie legt den Default fest. Die dritte Zeile gibt dem User »billg« Root-Zugang, er darf die Rechte anderer Leute ändern und andere administrative Aufgaben wahrnehmen. Die

vorletzten beiden Zeilen erlauben es allen Mitgliedern der Gruppe »developers«, den Entwicklungszeitweig »src/main« zu verändern und den Vendor-Zweig zu lesen. Zudem bekommt »billg« Schreib- (und darin eingeschlossen das Lese-) Recht für »vendor«.

Die letzte Zeile ist eigentlich überflüssig, denn »billg« hat als Superuser für alle Dateien bereits Schreibrechte darauf. Sie macht sich aber in dem Moment bezahlt, in dem der Superuser-Hut an jemand anderen übergeht.

Unantastbar

Nach diesem Ausflug in die Welt der Rechte nun zurück zum Buildmaster und seinem Coin-Upgrade. Während er sich mit Konflikten abmüht, ist eine Entwicklerin fleißig am Debuggen und verteilt jede Menge »printf« auf ihr »devel/coin«-Verzeichnis.

Wenn sie nun routinemäßig »p4 sync« aufruft, um die Änderungen ihrer Kollegen bei sich einzupflegen, updatet Perforce die meisten ihrer Coin-Dateien – jedoch nicht alle: Die Dateien mit dem Debug-Instrumentarium bleiben unangetastet. Solange man an einem File arbeitet, rührt »p4 sync« es nicht an. Die Entwicklerin kann entweder ihren Debug-Code mit »p4 revert« verwerfen oder

Listing 2: Ansicht für ein einfaches Projektmitglied

```
01 ...
02 # Use 'p4 help client' to see more about client
    views and options.
03 ...
04 Root:   /home/twee/work/projectx
05 ...
06 View:
07       //depot/projectx/src/main/... devel/...
```

Listing 3: Differenzierte Zugriffsrechte

```
01 Protections:
02     list user * * //...
03     super user billg * //...
04     write group developers * //depot/projectx/src/
    main/...
05     read group developers * //depot/projectx/vendor
    /...
06     write user billg * //depot/projectx/vendor/...
```

versuchen, ihn mit der neuen Version zu vereinen: »p4 resolve -am«.

In einer solchen Situation greift der Autor in der Regel zu »resolve«: Perforce löst die Merge-Probleme meist richtig und wenn am Ende doch eine Konfliktmeldung auftritt, bleibt immer noch die Entscheidung, ob man die Probleme von Hand lösen will oder sich in ein »p4 revert« flüchtet. Nach einem »revert« muss die Datei mit »p4 edit« wieder zum Schreiben geöffnet werden, um daran weiterzuarbeiten.

Etwas später ist der Bug gefixt – es ist also Zeit, die Änderungen einzuchecken. Vorsichtige beginnen mit »p4 diff -du« und schauen sich die Veränderungen im Unified-Diff-Format an. Jede Datei, für die das Diff lediglich Debug-Code enthält, ist revertierbar. Anschließend synchronisiert die Entwicklerin ihre Daten mit der neuesten Version, überprüft, ob alles läuft, schaut sich noch einmal sorgfältig das Diff an und ruft zu guter Letzt »p4 submit« auf.

Ist Perforce in andere Tools integriert (etwa »p4.el« für Emacs), sind andere Kommandos einzutippen, aber das Prinzip bleibt gleich.

Released

Irgendwann rückt der Release-Termin näher. Dann ist es das Einfachste, einen neuen Branch für die Release-Version anzulegen, den darin befindlichen Code zu testen und von Fehlern zu befreien, während die übrigen Entwickler weiter am Hauptentwicklungsstrang arbeiten. Allerdings sollen diese Fixes auch in den Hauptzweig integriert werden, damit sie in der übernächsten Ausgabe ebenfalls enthalten sind.

Einen Branch erzeugt »p4 integrate«; damit integriert man sozusagen alle Daten aus dem alten in einen neuen, noch leeren Zweig. Der Befehl erwartet als Argumente den zu kopierenden Zweig und das Ziel, den neuen Branch. Normalerweise handelt es sich bei beiden um Verzeichnisse.

Für die neue Release 1.1 erzeugt der Release-Manager »billg« einen neuen Ordner namens »rel1.1«, ruft »p4 client« auf und ändert seine Depot-Ansicht zu:

```
View:
//depot/projectx/src/main/... devel/...
//depot/projectx/vendor/... vendor/...
//depot/projectx/rel/1.1/... rel1.1/...
```

Anschließend integriert »billg« den in der Entwicklung befindlichen Sourcebaum nach »rel/1.1«, etwa mit:

```
cd ~/work/projectx
p4 integrate devel/... rel1.1/...
p4 submit rel1.1/...
```

Alternativ lassen sich die »p4«-Kommandos auch mit den Depot-Pfaden (»//depot/projectx/src/main/...« und so weiter) aufrufen.

In der Release-Phase arbeitet ein Teil des Teams am »rel/1.1«-Baum und checkt seine Korrekturen hier ein. Diese Bugfixes muss »billg« zurück in den Hauptzweig integrieren, damit auch die daran arbeitenden Kollegen davon profitieren. Dies geschieht mit:

```
p4 integrate rel1.1/... devel/...
```

Es ist dasselbe Kommando wie oben, nur dass sich die Integrationsrichtung umdreht. Treten Konflikte auf, etwa weil ein Workaround im Release-Zweig mit einer ordentlichen Lösung im Development-Branch kollidiert, wird eventuell ein »p4 resolve -am« nötig.

»integrate« kümmert sich dabei nur um jene Änderungen, die seit der letzten Integration neu hinzugekommen sind. Bei temporären Workarounds, die nur für diese eine Release gedacht sind, reicht es also, »p4« ein einziges Mal zu sagen, dass diese Änderung nicht integriert werden soll. Perforce wird sich beim nächsten Mal daran erinnern.

Wenn die Release-Version endlich fertig ist, kann man diesen Verzeichnisbaum

mit einem Label (»p4 label«) versehen. So wird es später einfacher, ein Diff zwischen Version 1.1 und 1.1.1 zu erstellen (»p4 diff« zeigt die Unterschiede zwischen lokalen Daten und den Daten im Depot; »p4 diff2« jene zwischen zwei Versionen im Depot) oder einen neuen Branch davon zu erzeugen, etwa wenn wegen einer Sicherheitslücke eine Version 1.1.0.1 nötig wird.

Weitere Features

Perforce kann noch eine Menge mehr. So verhilft der Perforce-Proxy »p4p« durch extensives Daten-Caching den auf mehrere Standorte verteilten Teams zu besserer Geschwindigkeit übers Netzwerk. Es gibt einen Hauptserver und an jedem Standort läuft ein Proxy. So kamen alle fünf Firmen, die weltweit an der Entwicklung von Unreal Tournament 2003 beteiligt waren, mit einem einzigen Perforce-Server aus.

Gerade bei der Spiele-Entwicklung, aber auch in allen anderen Bereichen, in denen Grafik- und andere Binärdateien versioniert werden müssen, macht sich Perforces Unterstützung für Dateitypen bemerkbar. Damit lässt sich zum Beispiel die Expansion des »\$Id\$«-Strings für XPM-Dateien abstellen. In einer 200 KByte großen Datei besteht eine 0,04 Prozent hohe Chance, eben diese vier Bytes »0x24 0x49 0x64 0x24«, also »\$Id\$«, zu finden. Werden XPM-Dateien wie (C++)-Code behandelt, ist die Wahrscheinlichkeit groß genug, dass ab und zu Dateien kaputtgehen, es sei denn, der SCM ist smart genug.

Perforce selbst enthält ein rudimentäres Jobtracking-System sowie ein Integrationssystem namens P4DTI. Letzteres integriert Perforce mit Bugzilla und ähnlichen Systemen und ist einen eigenständigen Artikel wert.

Wer sich nach einer Testphase dazu entschließt, Perforce für die Entwicklung freier Software einzusetzen, fragt bei [\[support@perforce.com\]](mailto:support@perforce.com) eine kostenlose Lizenz an und kann anschließend in Konkurrenz zu dem Autor treten: Dessen »p4d« musste nach 455 Tagen Up-time die Segel streichen – wegen eines Stromausfalls. (pju) ■

Information

PERFORCE
SOFTWARE

Perforce Software Europe

Yateley, UK

T: +44 1252 861400

F: +44 1252 861415

E-mail: info@perforce.com

URL: <http://www.perforce.com>